EBook Gratis

APRENDIZAJE mongoose

Free unaffiliated eBook created from
Stack Overflow contributors.

#mongoose

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: mongoose

It is an unofficial and free mongoose ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official mongoose.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con la mangosta

Observaciones

> **Mongoose** es una herramienta de modelado de objetos **MongoDB** diseñada para trabajar en un entorno asíncrono.
>
> Todo en Mongoose comienza con un esquema. Cada esquema se asigna a una colección MongoDB y define la forma de los documentos dentro de esa colección.

Mongoose hace que trabajar sin problemas con la base de datos MongoDB sea fácil.

Podemos estructurar fácilmente nuestra base de datos utilizando `Schemas and Models` , automatizar ciertas cosas cuando el registro se agrega o actualiza utilizando `Middlewares/Hooks` y obtener fácilmente los datos que necesitamos `querying` nuestros modelos.

Links importantes

- Mongoose Quickstart
- Mongoose GitHub Repository
- Documentación Mangosta
- Complementos Mangosta

Versiones

Último lanzamiento: Versión **4.6.0** publicada el **2 de septiembre de 2016**

Todas las versiones se pueden encontrar en
https://github.com/Automattic/mongoose/blob/master/History.md

Versión	Fecha de lanzamiento
1.0.1	2011-02-02
1.1.6	2011-03-22
1.3.0	2011-04-19
1.3.1	2011-04-27
1.3.4	2011-05-17
1.4.0	2011-06-10
1.5.0	2011-06-27
1.6.0	2011-07-07

Versión	Fecha de lanzamiento
2.0.0	2011-08-24
2.3.4	2011-10-18
2.5.0	2012-01-26
3.0.0	2012-08-07
3.1.2	2012-09-10
3.2.0	2012-09-27
3.5.0	2012-12-10
3.5.6	2013-02-14
3.6.0	2013-03-18
3.6.5	2013-04-15
3.8.0	2013-10-31
3.8.10	2014-05-20
3.8.15	2014-08-17
4.0.0	2015-03-25
4.0.6	2015-06-21
4.1.0	2015-07-24
4.2.0	2015-10-22
4.2.10	2015-12-08
4.3.5	2016-01-09
4.4.0	2016-02-02
4.4.4	2016-02-17
4.4.8	2016-03-18
4.4.13	2016-04-21
4.4.18	2016-05-21
4.5.0	2016-06-13

Versión	Fecha de lanzamiento
4.5.5	2016-07-18
4.5.8	2016-08-01
4.5.9	2016-08-14
4.5.10	2016-08-23
4.6.0	2016-09-02

Examples

Instalación

Instalar **mongoose** es tan fácil como ejecutar el comando `npm`

```
npm install mongoose --save
```

Pero asegúrese de que también haya instalado `MongoDB` para su sistema operativo o tenga acceso a una base de datos de MongoDB.

Conexión a la base de datos MongoDB:

1. Importar mangosta en la aplicación:

```
import mongoose from 'mongoose';
```

2. Especifique una biblioteca de Promesa:

```
mongoose.Promise = global.Promise;
```

3. Conectar a MongoDB:

```
mongoose.connect('mongodb://127.0.0.1:27017/database');

/* Mongoose connection format looks something like this */
mongoose.connect('mongodb://USERNAME:PASSWORD@HOST::PORT/DATABASE_NAME');
```

Nota:

- De forma predeterminada, la mangosta se conecta a MongoDB en el puerto `27017` , que es el puerto predeterminado utilizado por MongoDB.
- Para conectarse a MongoDB alojado en otro lugar, use la segunda sintaxis. Ingrese el nombre de usuario, contraseña, host, puerto y nombre de la base de datos de MongoDB.

El puerto de MongoDB es 27017 por defecto; Usa el nombre de tu aplicación como el nombre de la base de datos.

Conexión con opciones y callback.

Mongoose connect tiene 3 parámetros, uri, opciones y la función de devolución de llamada. Para usarlos ver muestra abajo.

```
var mongoose = require('mongoose');

var uri = 'mongodb://localhost:27017/DBNAME';

var options = {
    user: 'user1',
    pass: 'pass'
}

mongoose.connect(uri, options, function(err){
    if (err) throw err;
    // if no error == connected
});
```

Lea Empezando con la mangosta en línea:
https://riptutorial.com/es/mongoose/topic/1168/empezando-con-la-mangosta

Capítulo 2: Consultas de mangosta

Introducción

Mongoose es un controlador Node.JS para MongoDB. Proporciona ciertos beneficios sobre el controlador MongoDB predeterminado, como agregar tipos a los esquemas. Una diferencia es que algunas consultas de Mongoose pueden diferir de sus equivalentes de MongoDB.

Examples

Encontrar una consulta

Importe un modelo de Mongoose (consulte el tema "Esquemas de Mongoose")

```
var User = require("../models/user-schema.js")
```

El método findOne devolverá la primera entrada en la base de datos que coincida con el primer parámetro. El parámetro debe ser un objeto donde la clave es el campo a buscar y el valor es el valor que debe coincidir. Esto puede usar la sintaxis de consulta de MongoDB, como el operador de punto (.) Para buscar subcampos.

```
User.findOne({"name": "Fernando"}, function(err, result){
    if(err) throw err;    //There was an error with the database.
    if(!result) console.log("No one is named Fernando."); //The query found no results.
    else {
        console.log(result.name); //Prints "Fernando"
    }
}
```

Lea Consultas de mangosta en línea: https://riptutorial.com/es/mongoose/topic/9349/consultas-de-mangosta

Capítulo 3: Esquemas de mangosta

Examples

Esquema basico

Un esquema de usuario básico:

```
var mongoose = require('mongoose');

var userSchema = new mongoose.Schema({
    name: String,
    password: String,
    age: Number,
    created: {type: Date, default: Date.now}
});

var User = mongoose.model('User', userSchema);
```

Tipos de esquemas .

Métodos de esquema

Se pueden establecer métodos en los esquemas para ayudar a hacer cosas relacionadas con los esquemas y mantenerlos bien organizados.

```
userSchema.methods.normalize = function() {
    this.name = this.name.toLowerCase();
};
```

Ejemplo de uso:

```
erik = new User({
    'name': 'Erik',
    'password': 'pass'
});
erik.normalize();
erik.save();
```

Estática del esquema

Las estadísticas de esquema son métodos que pueden invocarse directamente por un modelo (a diferencia de los métodos de esquema, que deben invocarse mediante una instancia de un documento de Mongoose). Usted asigna una estática a un esquema agregando la función al objeto de `statics` del esquema.

Un ejemplo de caso de uso es para construir consultas personalizadas:

```
userSchema.statics.findByName = function(name, callback) {
    return this.model.find({ name: name }, callback);
}

var User = mongoose.model('User', userSchema)

User.findByName('Kobe', function(err, documents) {
    console.log(documents)
})
```

Esquema de geoobjetos

Un esquema genérico útil para trabajar con geo-objetos como puntos, cadenas de líneas y polígonos. Tanto **Mongoose** como **MongoDB son** compatibles con **Geojson** .

Ejemplo de uso en **Nodo / Express** :

```
var mongoose = require('mongoose');
var Schema   = mongoose.Schema;

// Creates a GeoObject Schema.
var myGeo= new Schema({
                        name: { type: String },
                        geo : {
                                type : {
                                         type: String,
                                         enum: ['Point', 'LineString', 'Polygon']
                                },
                                coordinates : Array
                         }
});

//2dsphere index on geo field to work with geoSpatial queries
myGeo.index({geo : '2dsphere'});
module.exports = mongoose.model('myGeo', myGeo);
```

Ahorro de tiempo actual y tiempo de actualización

Este tipo de esquema será útil si desea realizar un seguimiento de sus elementos por el tiempo de inserción o actualización.

```
var mongoose = require('mongoose');
var Schema   = mongoose.Schema;

// Creates a User Schema.
var user = new Schema({
                       name: { type: String },
                       age : { type: Integer},
                       sex : { type: String },
                       created_at: {type: Date, default: Date.now},
                       updated_at: {type: Date, default: Date.now}
});

// Sets the created_at parameter equal to the current time
user.pre('save', function(next){
    now = new Date();
```

```
    this.updated_at = now;
    if(!this.created_at) {
        this.created_at = now
    }
    next();
});

module.exports = mongoose.model('user', user);
```

Lea Esquemas de mangosta en línea: https://riptutorial.com/es/mongoose/topic/2592/esquemas-de-mangosta

Capítulo 4: Esquemas de mangosta

Examples

Creando un esquema

```
var mongoose = require('mongoose');

//assume Player and Board schemas are already made
var Player = mongoose.model('Player');
var Board = mongoose.model('Board');

//Each key in the schema is associated with schema type (ie. String, Number, Date, etc)
var gameSchema = new mongoose.Schema({
  name: String,
  players: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Player'
  }],
  host: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Player'
  },
  board: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Board'
  },
  active: {
    type: Boolean,
    default: true
  },
  state: {
    type: String,
    enum: ['decision', 'run', 'waiting'],
    default: 'waiting'
  },
  numFlags: {
    type: Number,
    enum: [1,2,3,4]
  },
  isWon: {
    type: Boolean,
    default: false
  }
});

mongoose.model('Game', gameSchema);
```

Lea Esquemas de mangosta en línea: https://riptutorial.com/es/mongoose/topic/6622/esquemas-de-mangosta

Capítulo 5: Mangosta Middleware

Observaciones

En mangosta, **Middlewares** también se llama como ganchos de `pre` y `post` .

Hay dos tipos de middleware

Ambos de estos middleware soportan ganchos **pre** y **post** .

1. **Documento middleware**

   Es compatible con las funciones de documento de `init` , `validate` , `save` y `remove`

2. **Middleware de consulta**

   Su soporte para las funciones de consulta `count` , `find` , `findOne` , `findOneAndRemove` , `findOneAndUpdate` , `insertMany` y `update` .

Ganchos pre y post

Hay dos tipos de ganchos **Pre**

1. **de serie**

   Como su nombre lo indica, se ejecuta en orden en serie i..e uno tras otro

2. **paralela**

   El middleware paralelo ofrece un control de flujo más preciso y el `hooked method` no se ejecuta hasta que el middleware paralelo llama a `done` .

Post Middleware se ejecuta después de que se haya completado el `hooked method` y todo su `pre` middleware.

Los métodos enganchados son las funciones soportadas por el middleware de documentos.
`init, validate, save, remove`

Examples

Middleware para hash contraseña de usuario antes de guardarlo

Este es un ejemplo de ***`Serial Document Middleware`***

En este ejemplo, escribiremos un middleware que convertirá la contraseña de texto sin formato en

una contraseña con hash antes de guardarla en la base de datos.

Este middleware se activará automáticamente cuando se cree un nuevo usuario o se actualicen los detalles de los usuarios existentes.

`User.js` : `User.js`

```
// lets import mongoose first
import mongoose from 'mongoose'

// lets create a schema for the user model
const UserSchema = mongoose.Schema(
  {
    name: String,
    email: { type: String, lowercase: true, requird: true },
    password: String,
  },
);


/**
 * This is the middleware, It will be called before saving any record
 */
UserSchema.pre('save', function(next) {

  // check if password is present and is modified.
  if ( this.password && this.isModified('password') ) {

    // call your hashPassword method here which will return the hashed password.
    this.password = hashPassword(this.password);

  }

  // everything is done, so let's call the next callback.
  next();

});


// lets export it, so we can import it in other files.
export default mongoose.model( 'User', UserSchema );
```

Ahora, cada vez que guardamos un usuario, este middleware verificará si la contraseña está **configurada** y **modificada** (esto es así, si la contraseña no se modifica), no tenemos hash.

`App.js` : `App.js`

```
import express from 'express';
import mongoose from 'mongoose';

// lets import our User Model
import User from './User';

// connect to MongoDB
mongoose.Promise = global.Promise;
```

```
mongoose.connect('mongodb://127.0.0.1:27017/database');


let app = express();
/* ... express middlewares here .... */


app.post( '/', (req, res) => {

  /*
    req.body = {
      name: 'John Doe',
      email: 'john.doe@gmail.com',
      password: '!trump'
    }
   */

  // validate the POST data

  let newUser = new User({
    name: req.body.name,
    email: req.body.email,
    password: req.body.password,
  });

  newUser.save( ( error, record ) => {
    if (error) {
      res.json({
        message: 'error',
        description: 'some error occoured while saving the user in database.'
      });
    } else {
      res.json({
        message: 'success',
        description: 'user details successfully saved.',
        user: record
      });
    }
  });

});



let server = app.listen( 3000, () => {
    console.log(`Server running at http://localhost:3000` );
  }
);
```

Lea Mangosta Middleware en línea: https://riptutorial.com/es/mongoose/topic/6646/mangosta-middleware

Capítulo 6: mangosta pre y post middleware (ganchos)

Examples

Middleware

El middleware (también llamado gancho previo y posterior) es una función a la que se le pasa el control durante la ejecución de funciones asíncronas. Middleware se especifica en el nivel de esquema y es útil para escribir complementos. Mongoose 4.0 tiene 2 tipos de middleware: middleware de documentos y middleware de consulta. El middleware de documentos es compatible con las siguientes funciones de documentos.

- en eso
- validar
- salvar
- retirar

El middleware de consulta es compatible con las siguientes funciones de modelo y consulta.

- contar
- encontrar
- Encuentra uno
- findOneAndRemove
- findOneAndUpdate
- actualizar

Tanto el middleware de documentos como el middleware de consulta admiten enlaces anteriores y posteriores.

Pre

Hay dos tipos de ganchos de pre, serie y paralelo.

De serie

El middleware serial se ejecuta uno tras otro, cuando cada middleware llama a continuación.

```
var schema = new Schema(..);
schema.pre('save', function(next) {
  // do stuff
  next();
});
```

Paralela

El middleware paralelo ofrece más control de flujo de grano fino.

var esquema = nuevo esquema (..);

```
// `true` means this is a parallel middleware. You **must** specify `true`
// as the second parameter if you want to use parallel middleware.
schema.pre('save', true, function(next, done) {
  // calling next kicks off the next middleware in parallel
  next();
  setTimeout(done, 100);
});
```

El método enganchado, en este caso, guardar, no se ejecutará hasta que lo haga cada middleware.

Publicar middleware

el middleware posterior se ejecuta después de que el método enganchado y todo su middleware anterior se hayan completado. el middleware posterior no recibe directamente el control de flujo, por ejemplo, no se le pasa ninguna devolución de llamada siguiente o realizada. Los ganchos de publicación son una forma de registrar escuchas de eventos tradicionales para estos métodos.

```
schema.post('init', function(doc) {
  console.log('%s has been initialized from the db', doc._id);
});
schema.post('validate', function(doc) {
  console.log('%s has been validated (but not saved yet)', doc._id);
});
schema.post('save', function(doc) {
  console.log('%s has been saved', doc._id);
});
schema.post('remove', function(doc) {
  console.log('%s has been removed', doc._id);
});
```

Lea mangosta pre y post middleware (ganchos) en línea:
https://riptutorial.com/es/mongoose/topic/4131/mangosta-pre-y-post-middleware--ganchos-

Capítulo 7: Población de mangosta

Sintaxis

1. Model.Query.populate (ruta, [seleccionar], [modelo], [coincidir], [opciones]);

Parámetros

Param	Detalles
camino	**Cadena** : la clave de campo que se va a rellenar
seleccionar	**Objeto, Cadena** - Selección de campo para la consulta de población.
modelo	**Modelo** - Instancia del modelo de referencia
partido	**Objeto** - Condiciones de poblado
opciones	**Objeto** - Opciones de consulta

Examples

Poblado simple

Mongoose populate se usa para mostrar datos de documentos de referencia de otras colecciones.

Digamos que tenemos un modelo de `Person` que tiene documentos de referencia llamados `Address` .

Modelo de persona

```
var Person = mongoose.model('Person', {
    fname: String,
    mname: String,
    lname: String,
    address: {type: Schema.Types.ObjectId, ref: 'Address'}
});
```

Modelo de dirección

```
var Address = mongoose.model('Address', {
    houseNum: String,
    street: String,
    city: String,
    state: String,
    country: String
});
```

Para rellenar la `Address` dentro de la `Person` usa su ObjectId, usando digamos `findOne()` , use la función `populate()` y agregue la `address` clave del campo como primer parámetro.

```
Person.findOne({_id: req.params.id})
    .populate('address') // <- use the populate() function
    .exec(function(err, person) {
        // do something.
        // variable `person` contains the final populated data
    });
```

O

```
Person.findOne({_id: req.params.id}, function(err, person) {
    // do something
    // variable `person` contains the final populated data
})
.populate('address');
```

La consulta anterior debe producir el documento a continuación.

Persona doc

```
{
    "_id":"123abc",
    "fname":"John",
    "mname":"Kennedy",
    "lname":"Doe",
    "address":"456def" // <- Address' Id
}
```

Dirección doc

```
{
    "_id":"456def",
    "houseNum":"2",
    "street":"Street 2",
    "city":"City of the dead",
    "state":"AB",
    "country:"PH"
}
```

Doc poblado

```
 {
    "_id":"123abc",
    "fname":"John",
    "mname":"Kennedy",
    "lname":"Doe",
    "address":{
        "_id":"456def",
        "houseNum":"2",
        "street":"Street 2",
        "city":"City of the dead",
        "state":"AB",
        "country:"PH"
```

```
    }
}
```

Descuidar algunos campos

Supongamos que **no desea que** los campos `houseNum` y `street` en el campo de `address` del documento final finalizado, utilice el `houseNum` `populate()` siguiente manera:

```
Person.findOne({_id: req.params.id})
    .populate('address', '-houseNum -street')  // note the `-` symbol
    .exec(function(err, person) {
        // do something.
        // variable `person` contains the final populated data
    });
```

O

```
Person.findOne({_id: req.params.id}, function(err, person) {
    // do something
    // variable `person` contains the final populated data
})
.populate('address', '-houseNum -street');  // note the `-` symbol
```

Esto producirá el siguiente documento final completado,

Doc poblado

```
 {
    "_id":"123abc",
    "fname":"John",
    "mname":"Kennedy",
    "lname":"Doe",
    "address":{
        "_id":"456def",
        "city":"City of the dead",
        "state":"AB",
        "country:"PH"
    }
}
```

Rellenar solo unos pocos campos

Si **solo desea que** los campos `houseNum` y `street` en el campo de `address` en el documento final completado, use la función `populate()` siguiente manera en los dos métodos anteriores,

```
Person.findOne({_id: req.params.id})
    .populate('address', 'houseNum street')
    .exec(function(err, person) {
        // do something.
        // variable `person` contains the final populated data
    });
```

O

```
Person.findOne({_id: req.params.id}, function(err, person) {
    // do something
    // variable `person` contains the final populated data
})
.populate('address', 'houseNum street');
```

Esto producirá el siguiente documento final completado,

Doc poblado

```
 {
    "_id":"123abc",
    "fname":"John",
    "mname":"Kennedy",
    "lname":"Doe",
    "address":{
        "_id":"456def",
        "houseNum":"2",
        "street":"Street 2"
    }
}
```

Población anidada

Digamos que tiene un esquema de `user` , que contiene `name` , `contactNo` , `address` y `friends` .

```
var UserSchema = new mongoose.Schema({
    name : String,
    contactNo : Number,
    address : String,
    friends :[{
        type: mongoose.Schema.Types.ObjectId,
        ref : User
    }]
});
```

Si desea encontrar un usuario, sus **amigos** y **amigos de amigos** , necesita hacer población en 2 niveles, es decir, **Población anidada** .

Para encontrar amigos y amigos de amigos:

```
User.find({_id : userID})
    .populate({
        path : 'friends',
        populate : { path : 'friends'}//to find friends of friends
    });
```

Todos los `parameters` y `options` de `populate` se pueden usar también dentro del poblado anidado, para obtener el resultado deseado.

Del mismo modo, puede `populate` más `levels` acuerdo a sus requerimientos.

No se recomienda hacer población anidada por más de 3 niveles. En caso de que necesite hacer

un relleno anidado para más de 3 niveles, es posible que deba reestructurar su esquema.

Lea Población de mangosta en línea: https://riptutorial.com/es/mongoose/topic/2616/poblacion-de-mangosta

Capítulo 8: Población de mangosta

Sintaxis

- Query.populate (ruta, [seleccionar], [modelo], [coincidir], [opciones])

Parámetros

Parámetro	Explicación
camino	< **Object, String** > ya sea la ruta para rellenar o un objeto que especifica todos los parámetros
[seleccionar]	< **Objeto, Cadena** > Selección de campo para la consulta de población (puede usar `'-id'` para incluir todo menos el campo de `id`)
[modelo]	< **Modelo** > El modelo que desea utilizar para la población. Si no se especifica, poblar buscará el modelo por el nombre en el campo de referencia del esquema.
[partido]	< **Objeto** > Condiciones para la población
[opciones]	< **Objeto** > Opciones para la consulta de población (ordenación, etc.)

Examples

Un simple ejemplo de mangosta

`.populate()` en Mongoose le permite llenar una referencia que tenga en su colección o documento actual con la información de esa colección. Lo anterior puede parecer confuso, pero creo que un ejemplo ayudará a aclarar cualquier confusión.

El siguiente código crea dos colecciones, Usuario y Publicación:

```
var mongoose = require('mongoose'),
  Schema = mongoose.Schema

var userSchema = Schema({
  name: String,
  age: Number,
  posts: [{ type: Schema.Types.ObjectId, ref: 'Post' }]
});

var PostSchema = Schema({
  user: { type: Schema.Types.ObjectId, ref: 'User' },
  title: String,
  content: String
```

```
});

var User = mongoose.model('User', userSchema);
var Post = mongoose.model('Post', postSchema);
```

Si quisiéramos rellenar todas las publicaciones para cada usuario cuando `.find({})` todos los Usuarios, podríamos hacer lo siguiente:

```
User
  .find({})
  .populate('posts')
  .exec(function(err, users) {
    if(err) console.log(err);
    //this will log all of the users with each of their posts
    else console.log(users);
  })
```

Lea Población de mangosta en línea: https://riptutorial.com/es/mongoose/topic/6578/poblacion-de-mangosta

Creditos

S. No	Capítulos	Contributors
1	Empezando con la mangosta	4444, CENT1PEDE, Community, Delapouite, dublicator, jisoo, Random User, zurfyx
2	Consultas de mangosta	Gibryon Bhojraj
3	Esquemas de mangosta	AndreaM16, Ian, zurfyx
4	Mangosta Middleware	Delapouite, Random User
5	mangosta pre y post middleware (ganchos)	Naeem Shaikh
6	Población de mangosta	CENT1PEDE, Chinni, Medet Tleukabiluly, Ravi Shankar